Appendix 1

MISRA-C導入のススメ

～実運用のためのQ＆A～

編集部

C言語における「危険なプログラミング」を避けるため，「MISRA-C（正式名称は“Guidelines for the Use of the C Language in Vehicle Based Software”）」というコーディング規約が自動車業界で注目されている．ここでは，MISRA-Cに造けいが深いエンジニアの方々に，こうしたコーディング規約の必要性，また実設計に生かすための考えかたなどを聞いた．聞き手は本誌編集部．（編集部）

【答えていただいた方々】

岩崎 保氏：NECエレクトロニクス．MISRA-Cを含む車載関連のソフトウェア開発環境の顧客サポートに従事．JasParの活動に参加

舘 伸幸氏：NECマイクロシステム．組み込みソフトウェア開発者．MISRA-C準拠の車載ソフトウェアの開発を経験

中村さおり氏：NECエレクトロニクス．ソフトウェア開発ツール担当．MISRA-C研究会[注1]に参加

吉澤智美氏：NECエレクトロニクス．ソフトウェア開発ツール担当．「組込みソフトウェア管理者・技術者育成研究会（SESSAME）」に参加

Q1：車載ソフトウェア設計者にMISRA-Cは必須ですか？

岩崎氏：規約に精通しておけとは言いませんが，少なくとも「MISRA-Cって何？」などと言わない程度にはなっていてほしいです．

舘氏：私が開発した車載ソフトウェアについて言えば，「ソース・コードはMISRA-C準拠か」という問い合わせや，開発の請負時に「MISRA-Cもしくはそれに相当するコーディング規約に準拠してほしい」という要求が近年は顕著です．

Q2：ルールをすべて覚えてから設計するものですか？

舘氏：基本的にはそうです．ただし，MISRA-Cの項目[注2]を一字一句覚えるというわけではありません．MISRA-Cをもとに社内でコーディング規約を作り，それに目を通しておきます．規約を読んでみると経験的に納得できることが多く，そんなに苦労しなくてもなんとなく頭に入るものです．ある程度規約を理解したうえでコーディングし，チェッカを通してみて，そこでだめな記述のしかたがわかれば，それを踏まえて後2，3回コーディングしていけば自然と身につきます．

Q3：作業効率は落ちませんか？

舘氏：コーディングだけを見ると，確かに（ルールに合っているかどうかの）チェックの分だけ手間はかかります．ただし，コーディング規約によって防げるバグというのは少なくありません．そもそもコーディング規約は二つ目的があって，ケアレス・ミスをなくすこと，そして可読性の向上です．ケアレス・ミスによるバグほど見つかりにくいのです．また，記述がバラバラだと読みにくく，レビュー効率が落ちます．ある程度記述がそろっていれば他人のコードでも腹を立てずに（笑）読めるようになります．結果として，コーディングには確かに時間はかかりますが，開発の全工程で見ると効率は上がると思います．

Q4：「MISRA-C準拠」とは「すべてのルールを絶対に守る」という意味ですか？

中村氏：MISRA-Cのルールは，すべてを絶対に守らないとい

注1：MISRA-C研究会は，MISRA-Cを実設計で使うための研究を行っている．同研究会の検討結果などをまとめたものが，解説書「組込み開発者におくるMISRA-C―組込みプログラミングの高信頼化ガイド」（日本規格協会 刊）として発売されている．

注2：MISRA-C:1998では127項目のルールが，MISRA-C:2004では141項目のルールが定められている．これらの日本語訳は自動車技術会からテクニカル・ペーパとして入手可能．購入方法などについては，同技術会のWebサイト（http://www.jsae.or.jp/）を参照．

KeyWord　MISRA-C，コーディング規約，C言語，SESSAME，MISRA-C研究会，可読性，保守性，テーラリング，逸脱手順，チェッカ

【MISRA-Cの項目のタイプ】

1）かならず仕立て（テーラリング）が必要なもの

【例】項目No.13「char，int，long，floatおよびdouble という基本型は使用しないこと．代わりに個々のコンパイラに対して特定長の同等物をtypedef したうえでコードではこれらが使用されるべきである」

【解説】組み込みソフトウェアでは変数のビット幅（バイト幅）が重要な意味を持つことが多いので，ひとめでそれがわかるものが望ましい．また，変数名そのものにも同様の型名を付けることでその変数のサイズを明示することができる．このため，なるべく短い名称で統一性のあるものが求められる．一般によく使われるのは，signed/unsigned を区別する1文字と，その後にビット幅かバイト幅の数値を組み合わせる方法．前者においてcharはs8となり，unsigned shortはu16となる．後者は同様にs1，u2となる．

2）逸脱の可否を単純には判断できないもの

【例】項目No.58「（switch文の条件を終了する以外は）break文は使用してはならない」

【解説】逸脱することで可読性が向上する場合がある．多くの場合，プログラム構造をよく考えればbreakやcontinueというみっともない制御文は入れずにすむ．ただし例外もある（本誌付属CD-ROM収録の記述を参照）．自組織のコーディング規約の策定では，あくまでMISRA-C遵守とするか，規約は選択可能としておきレビュー項目でその逸脱の妥当性を検討することを必須とするなどの方法が考えられる．

3）単に守るだけでは不十分なもの

【例】項目No.111「ビット・フィールドは、unsigned int型かsigned int型に対してのみ定義しなければならない」

【解説】マイコン専用のCコンパイラでは，内蔵ペリフェラルの制御レジスタのビット割り当てを，unsigned charをベースにしたビット・フィールド構造体としているものがある．この場合，MISRA-C規定を優先させて書き換えると逆に品質を落とすことになりかねない．また，ビット・フィールドの操作にはリード・モディファイ・ライトのコードが生成される場合がある．これは，リードからライトの間に割り込みが入り，対象ワードに含まれる別ビットの操作が行われても，割り込みから戻った直後のライトで上書きされてしまうというバグにつながる．No.111を守りさえすれば安全とはいえない．

4）少し拡張して考えておくとよいもの

【例】No.84「voidの戻り値を持つ関数では，return文が式を持ってはならない」

【解説】これは当然であり，ほとんどのコンパイラはエラーを検出してくれる．しかし，もう少し拡張して考えて，「void関数では式を持たないreturn文を明記する」というルールを作ることもできる．関数も表記パターンを統一できるだけでなく，「ここは後で続きを書こう」として失念したままビルドしてしまうといった事故を，少なくともコード・レビューなどで発見しやすくすることができる．

図1　MISRA-Cの項目のタイプの例

MISRA-Cの項目にはいくつかのタイプがある．実際の現場では，実装は扱うシステムや設計手法に大きく依存する．したがって，MISRA-Cの「知恵」を基本に，組織に合った規約を作成し，運用するという姿勢が現実的

けないということはありません．守っていると逆に（プログラムの）品質が落ちかねないようなルールもないとは言えません．また，可読性という面でも問題になります．そのため，MISRA-Cでも場合によってはルールを逸脱[注3]してもよいということになっています．

吉澤氏：でも，「逸脱してはならない」と悩んでいる人もいるみたいです．規約にのっとることを重視して，「逸脱しないとするとかえってソフトウェアの品質や保守性が落ちるけど，どうすればいいんだろう」と．

中村氏：それは逆にMISRAの精神に反することになります．正しい逸脱手続きを踏めば（理由が明確ならば），かならずしも規約を守る必要はないのです．

舘氏：また，実際にMISRA-Cを使おうとするとテーラリングが必要です（**図1**）．つまり，MISRA-Cをそのまま使うのではなく，それをベースに“きちんと”社内でコーディング規約を作らないといけないと思います．社内でコーディング規約を作るときにMISRA-Cのルール（逸脱）の矛盾を，ある程度吸収できます．「こっちのルールには引っかかるけど，こっちのルールを優先しようね」とか．

注3：MISRA-Cでは逸脱する際に「各個人で決めない」，「逸脱理由を文書として残す」などとしているが，具体的な手順を決めているわけではない．

注4：組込みソフトウェア管理者・技術者育成研究会（SESSAME）のWebサイト（http://www.sessame.jp/）にある「ハードウェア出身のマネージャに分かっておいてほしい7つのこと」を参照．

Q5：「正しい逸脱手順」とは？

舘氏：私のところでは，（コーディングしたプログラムがMISRA-Cのチェッカに）引っかかったら，どういうワーニングが出たか，それに対してどういう処置（逸脱手続き）をとったか，みんなでレビューしています．

中村氏：まっとうな手続きですね．逸脱理由は文書として（ソフトウェアの発注側に）提出するのですか？

舘氏：それは設計書など（必要に応じて逸脱理由単体の文書）に記載したりします．

岩崎氏：ソフトウェアを作った側から「ここを逸脱しました」と報告する場合や，発注する側が発注時に「この部分はこういうふうに逸脱してもかまわない」と規定してくる場合があると思います．どのような形で逸脱理由を提出するかは，発注元のメーカの考えかたによるのではないでしょうか．

吉澤氏：「正しい逸脱手続きを踏む」というところはマネージメントの問題になります．逸脱手続きをどうするかをきちんと決めておかないと，（実際にコーディングを行う技術者が）苦労することになります．

舘氏：品質について突き詰めると，結局そのプロジェクト・マネージャが「ソフトウェアの品質」についてどう考えているかというところに行き着くと思います．そういった意味で，マネージャの方にはソフトウェア開発のことを少しでも知っておいていただきたいです[注4]．